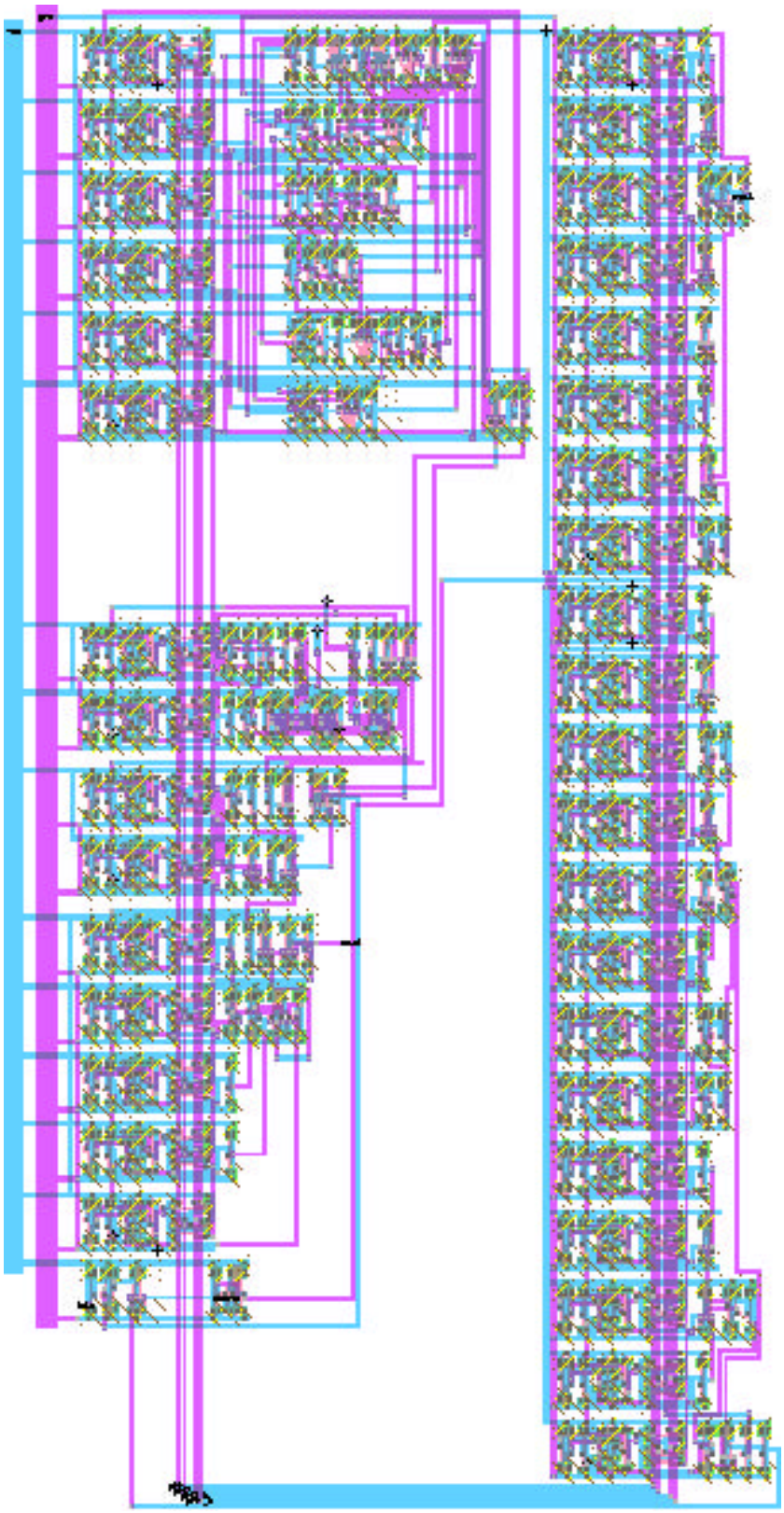


PlayStation Mod Chip

Mike Aung
Matthew Gay

E 158

April 11, 2001

Introduction

The *Sony PlayStation Computer Entertainment System* is a video game system marketed all over the world by Sony. For marketing and sales purposes, there are three different versions of the system sold in separate geographical regions: Japan, Europe, and the U.S. Unfortunately, the games are also created in different versions, so that a game purchased in Japan cannot be played on a PlayStation purchased in the U.S. This is done for marketing purposes as well as to allow games to be priced differently for different regions. To work around this problem, a gray market of "mod chips" has developed. A mod chip is a device that is soldered onto the PlayStation motherboard in such a way that the PlayStation will play games from any of the three regions regardless of its country of origin. These mod chips also allow the PlayStation to play pirated games that have been copied using a CD-ROM burner.

Most PlayStation owners obtain mod chips through commercial retailers who install the chips for you. It is also possible to construct your own mod chip, and there are sites on the web describing the operation of these chips. Most of these chips use some sort of simple microprocessor programmed to act like a mod chip. A good example of this can be found at http://elm-chan.org/reports/psm/ where they use an Atmel microcontroller as the mod chip. For this project, we constructed a functioning mod chip using Electric to lay out the chip.

Functional Overview

When a game disc is inserted into the PlayStation, the machine checks for a region code on the disc during the boot sequence. A mod chip tricks the PlayStation into believing that any game is an in-region game by blocking the region code from the CD, and replacing it with the correct region code. The mod chip does this by outputting all of the region codes one after another. Originally the region codes were output as long as the machine was turned on, but game developers figured this out and started checking to see if the region codes were being output at times other than start-up, and then stopping the game. To counter this, "stealth" mod chips were released, which stopped outputting the region codes after a set period of time, or after a few other events, which implied that the game had already checked the region code. To make our chip as versatile as possible, and to add some challenge to the project, we also plan to make our chip a stealth mod chip.

The codes that are output consist of: SCEI (Japan), SCEA (America), SCEE (Europe). These codes are output in ASCII one after another with a 72ms delay in between codes until one of the following conditions are met:

- Timing out – after 26.5 seconds, the region codes are no longer sent
- Memory card access – since most games access the memory card before checking for a mod chip
- Cover open – when the cover of the PlayStation is open, the sending of the region codes is suspended
- Reset – when the reset button is released, the chip starts over in sending the region codes.

The reset, memory check, and cover open signals are all supplied by the PlayStation, so all we have to do is pull them off of the board and supply them to the chip.

In hex, the bitstream out of the chip should be:

9A93D2BA5B4 (Japan)
9A93D2BA574 (Europe)
9A93D2BA5F4 (USA)
…

This translates to:

1001 1010 1001 0011 1101 0010 1011 1010 0101 1011 0100
1001 1010 1001 0011 1101 0010 1011 1010 0101 0111 0100
1001 1010 1001 0011 1101 0010 1011 1010 0101 1111 0100
…

There should be a 72ms delay between each code, and each bit is asserted for 4 ms.

After struggling for a while, we decided that we were unable to implement the chip in a way that cycled through the 3 different countries on its own. Instead, we have a two bit input corresponding to the country. This doesn't really reduce the functionality of the mod chip, because for any given PlayStation the chip only needs to output one country code. So, when the chip is installed in a PlayStation, the con_code inputs can be hard-wired to the appropriate values. This changes the expected output of the chip to (for Japan):

1001 1010 1001 0011 1101 0010 1011 1010 0101 1011 0100
…

Chip Pinout

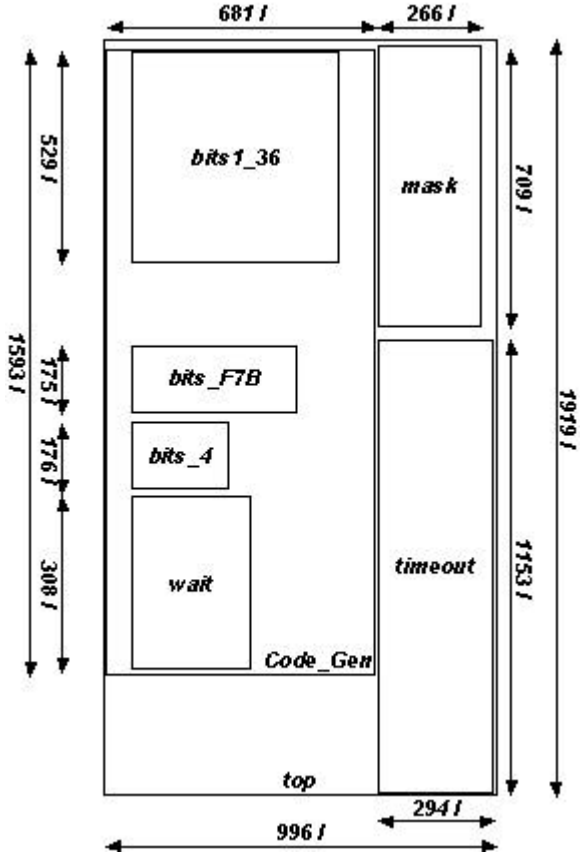
Since we don't plan for our chip to be fabricated, we don't have it attached to a pad frame. However, we do have the following inputs and outputs:

phi1	Clock	Phase 1, 250 Hz clock
phi1_b	Clock	Not phi1
phi2	Clock	Phase 2, 250 Hz clock
phi2_b	Clock	Not phi2
reset	Input	Reset signal from PlayStation (active high)
cover	Input	Cover open signal from PlayStation (active high)
mem	Input	Memory access signal from PlayStation (active high)
con_code0	Input	These are used to tell the chip which version PlayStation it is installed in (00 = Europe, 01 = Japan, 10 = USA).
con_code1	Input	
mask	Output	Blocks the real data from the disk by driving it low.
fake	Output	The fake data provided by the chip to the PlayStation

Chip Floorplan

Below is the floorplan of our final design. It manages to fit nicely into the 2000λ x 2000λ space available on a Tiny chip. We were pleasantly surprised to discover that our original estimates for

the dimensions of the various facets were fairly accurate. All of the facets turned out to be close to (but not larger than) the estimated size.



Area and Time Data

As we worked on the project, we kept track of the time required for each facet, as well as the area of each facet. This data is shown below:

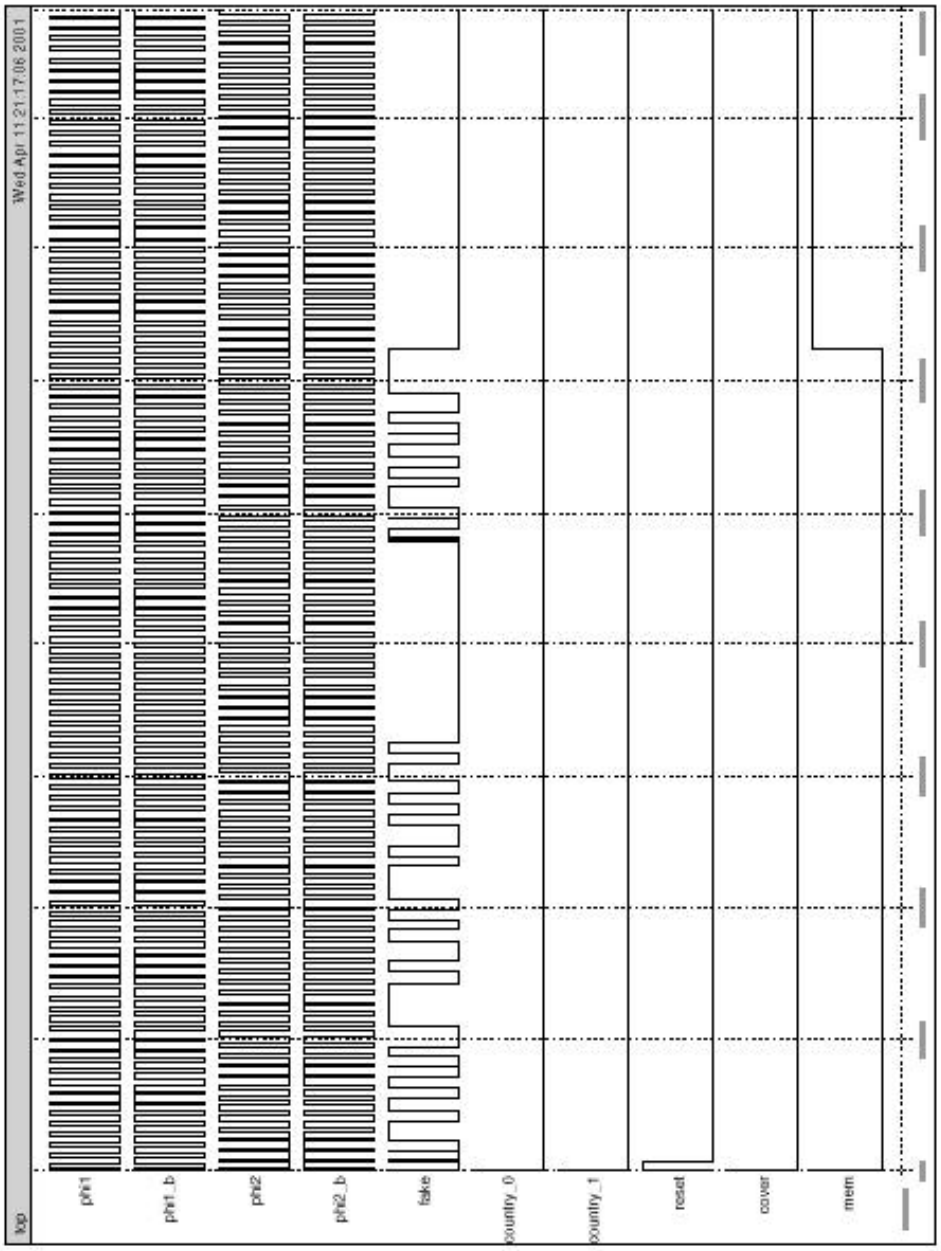
Cell	Time (hrs)	Width(l)	Height(l)	Area(l^2)
Top	4	996	1918.5	1,910,826
Mask	2	266.375	709	188,860
Timeout	2	293.5	1153	338,406
Code_Gen	6	681	1592.5	1,084,493
bits1_36	2.5	529.25	537	284,207
36bits	5	347	537	186,339
bits_F7B	3	420	175	73,500
bits_4	2	249.5	174.5	43,538
wait	2	307.5	441	135,608
con_code	2	240.5	174	41,847
2-phaselatch	1	72.5	87	6,308
half_adder	1	106.5	87	9,266
counter	4	180.25	87	15,682
counter_2	1	191	174	33,234
counter_5	1	193.5	441	85,334
counter_6	1	188.5	537	101,225

counter_8	1	190.5	709	135,065
counter_13	1	190.5	1153	219,647
tri	0.25	30.5	87	2,654
mux2	0.25	52	87	4,524
mux4	0.25	183	88	16,104
inv	0.25	25	87	2,175
nand2	0.25	33	87	2,871
std_nand3	0.25	47	87	4,089
and2	0.25	52	87	4,524
and3	0.25	64	87	5,568
and4	0.25	83.5	87	7,265
and5	1	114	88	10,032
and6	0.25	110	87	9,570
xor2	5	60	87	5,220
Std_nor2	0.25	34.5	87	3,002
Std_nor3	0.25	41	87	3,567
or2	0.25	52	87	4,524
or3	0.25	57.25	87	4,981
or4	0.25	83	87	7,221
or6	0.25	98	87	8,526
totals:	51.5			4,999,795

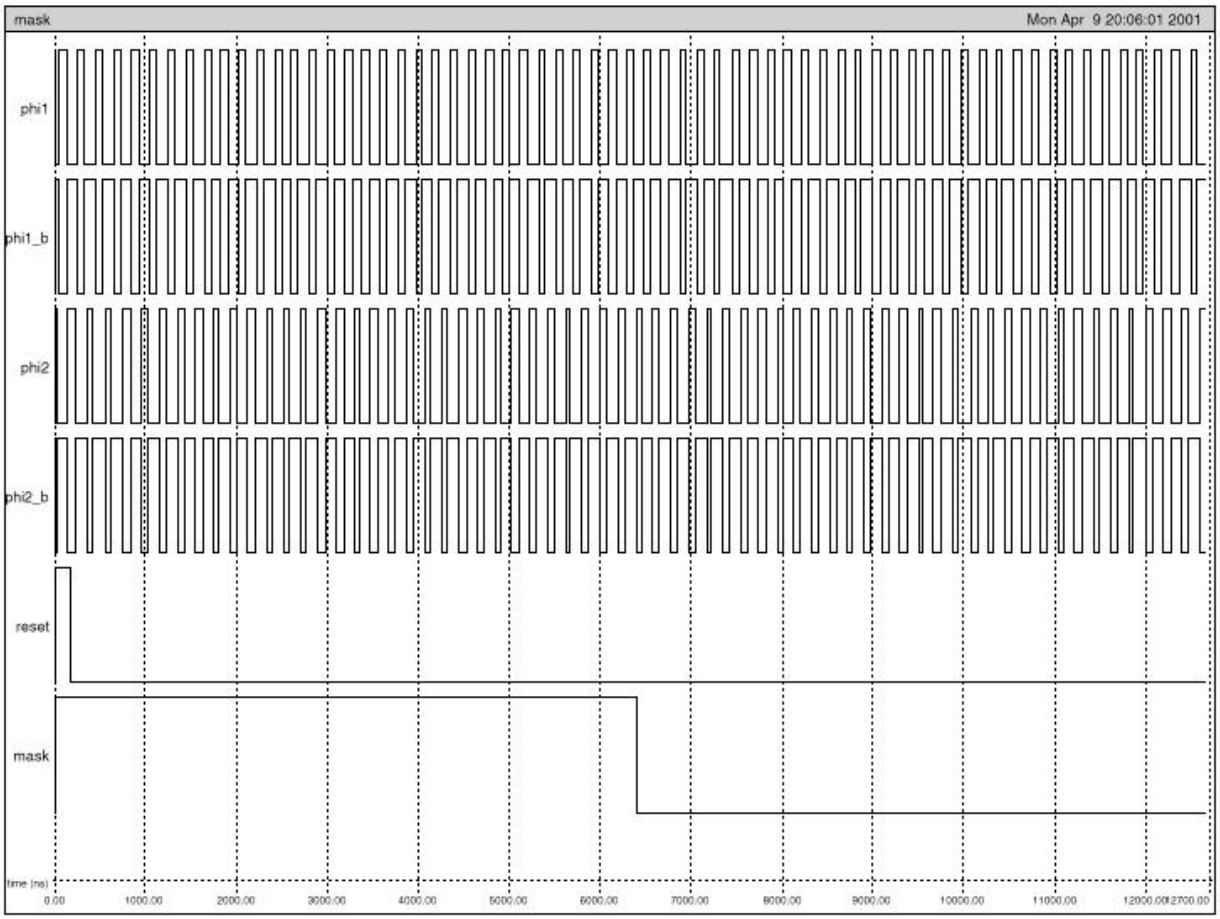
The time spent on the design seemed to be spent mostly on drawing and debugging layouts. At first a significant amount of time was spent on getting layouts to pas NCC, but after a while, we developed a systematic approach to fixing NCC errors, and became more efficient.

Simulation Results

We got the top level schematic to simulate with some success. As long as the country code is 00 (Europe), the simulation works fine. However, when the country code is switched, the fake output goes permanently high. This is because of a poorly designed reset in the bits_F7B block. When the block is reset, the counters output 00. This means that the fake output of the block outputs whatever the first bit of that country code is. In the case of Europe, the output is 0111, so the first bit output is 0, which is not a problem. For Japan and USA however, the outputs are 1011 and 111 respectively. In these cases, the first output bit is 1. This means that when the chip is reset, the bits_F7B block outputs 1 while it's waiting to start counting. This overrides the outputs of all the other blocks. This can be fixed by taking the fake output and "anding" it with the cin input. That way, the output of the block will be zero unless it's actually in the process of counting. Apart from the difficulty with the country bits, each of the individual parts of the chip do simulate as we expected.



The simulation of the top layer (shown above) demonstrates that it is able to output the necessary codes with the appropriate 72 ms wait in between (note that the plot is not to scale). Then, the output drops to zero as soon as the mem signal is driven high. This is what we expect the chip to do.



The mask block outputs the signal that covers up the real data from the PlayStation. This can be seen above as the mask output drops to low, covering up the real data, after a start-up time.

The timeout block would be simulated to demonstrate that the chip stops sending data after 26.5 seconds, however, IRSIM has a time limit, and it is impossible to simulate the block for enough clock cycles.

Verification Results

For each chip facet, we ran a set of tests to make sure that our layouts were correct. First, we ran a Design Rules Check (DRC) to make sure that we hadn't violated any of the desisign rules. Next we ran an Electrical Rules Check (ERC). This checked that all of the wells were sufficiently grounded or powered. Finally we ran a Network Compare Check (NCC) to compare the schematic to the layout. There were three different versions of Network compare: check current facets only, flatten hierarchy, and recursively check subfacets. We ran all three versions for each facet, and made sure that it passed each. Once DRC, ERC, and NCC had been passed, we were reasonably certain that the layout could be manufactured and matched the schematics. The chart below shows each facet and its status on the tests (everything passed):

Cell	Schematic	Layout	DRC	ERC	NCC
top	X	X	X	X	X
mask	X	X	X	X	X
timeout	X	X	X	X	X
Code_Gen	X	X	X	X	X
bits1_36	X	X	X	X	X
36bits	X	X	X	X	X
bits_F7B	X	X	X	X	X
bits_4	X	X	X	X	X
wait	X	X	X	X	X
con_code	X	X	X	X	X
2-phaselatch	X	X	X	X	X
half_adder	X	X	X	X	X
counter	X	X	X	X	X
counter_2	X	X	X	X	X
counter_5	X	X	X	X	X
counter_6	X	X	X	X	X
counter_8	X	X	X	X	X
counter_13	X	X	X	X	X
tri	X	X	X	X	X
mux2	X	X	X	X	X
mux4	X	X	X	X	X
inv	X	X	X	X	X
nand2	X	X	X	X	X
and2	X	X	X	X	X
and3	X	X	X	X	X
and4	X	X	X	X	X
and5	X	X	X	X	X
and6	X	X	X	X	X
xor2	X	X	X	X	X
nor2	X	X	X	X	X
nor3	X	X	X	X	X
or2	X	X	X	X	X
or3	X	X	X	X	X

or4	X	X	X	X	X
or6	X	X	X	X	X

Test Plan

The obvious way to test this chip is to plug it into a PlayStation. However, we are very hesitant to actually do this. There are a few reasons we are concerned. First of all, installing the chip would involve soldering directly to the machine, and we really don't want to risk a $100 piece of equipment. The other reason that we are hesitant to plug into the PlayStation is that even good mod chips aren't perfect. Some games work well with some mod chips, so if the chip failed to work, it would be hard to tell if the failure was due to the mod chip, or due to some unique feature of the games we were using. Also, different versions of the PlayStation require slightly different versions of the mod chip.

Rather than testing the chip on an actual PlayStation, we will test it using a logic analyzer. The chip should be plugged into the appropriate clock signals, the outputs mask and fake will be connected to the logic analyzer. The test sequence will be as follows:

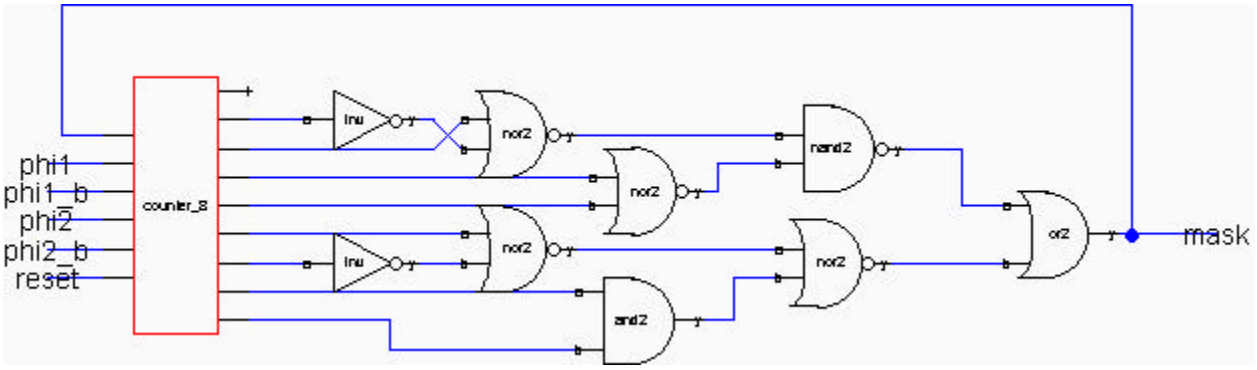
1. Power up the chip with reset high, and all other inputs low
2. Set reset low
3. Check that mask goes low after 900 ms (225 clock cycles)
4. Watch the bitstream out of fake, and check that it is as described in the functional overview of the chip (make certain that it is repeating itself).
5. Do one of the following:
    a. Wait 26.5 s (6625 clock cycles) and check that the fake output returns to low
    b. Drive the mem input high and check the fake output immediately drops to low
    c. Drive the cover input high and check the fake output immediately drops to low
6. Drive reset high for a few cycles (the time doesn't matter since in actual usage, no user can push reset in less than 4 ms), then return to step 2, selecting a different option at step 5.
7. Finally, leave the chip on for a few minutes to ensure that the fake output stays low rather than returning high, and to check for any other unexpected behavior.

This should test all the functions of the chip. If satisfied that the chip is behaving as expected, it can be plugged into an actual PlayStation. The wiring diagrams for mod chips are easily found on the Internet. Before actually plugging the chip in, it would be wise to check the signals coming to the various inputs (mem, cover, and reset). There is some ambiguity as to whether they are active high or active low. The chip is expecting active high inputs, so if the PlayStation is providing active low inputs, it may be necessary to run them through inverters.

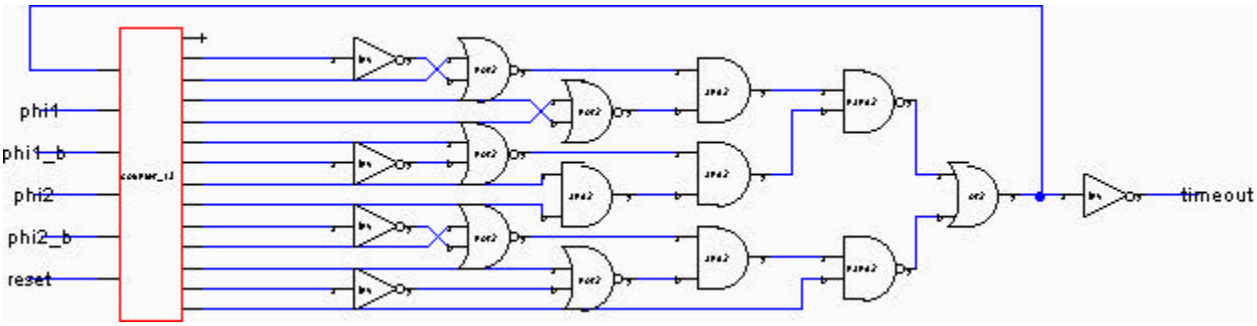
Schematics

top

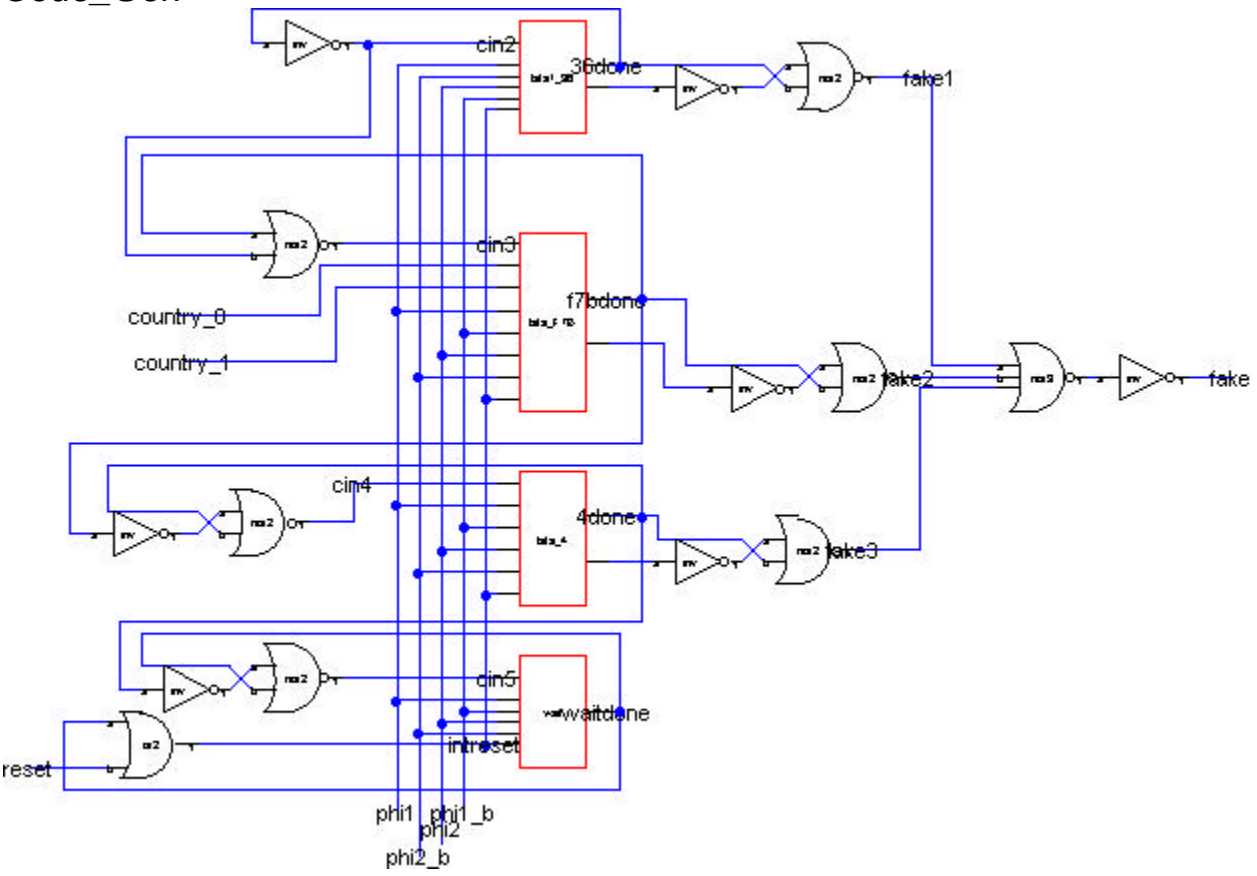
mask



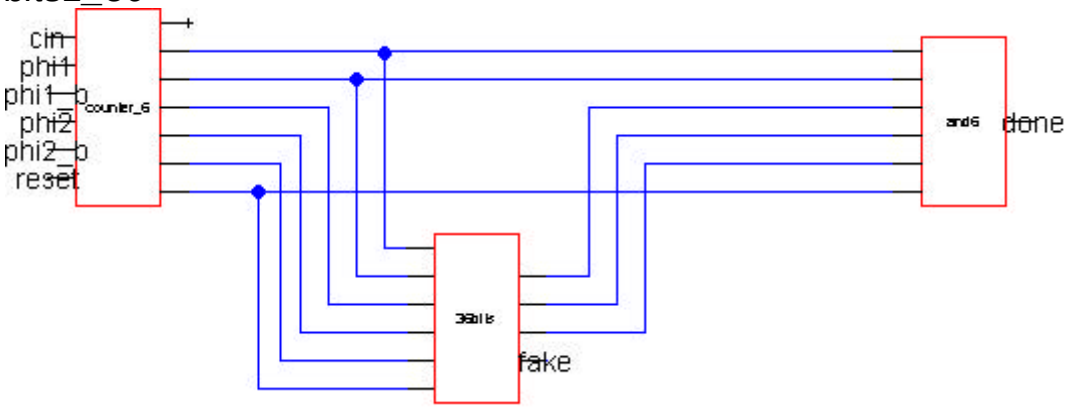
timeout



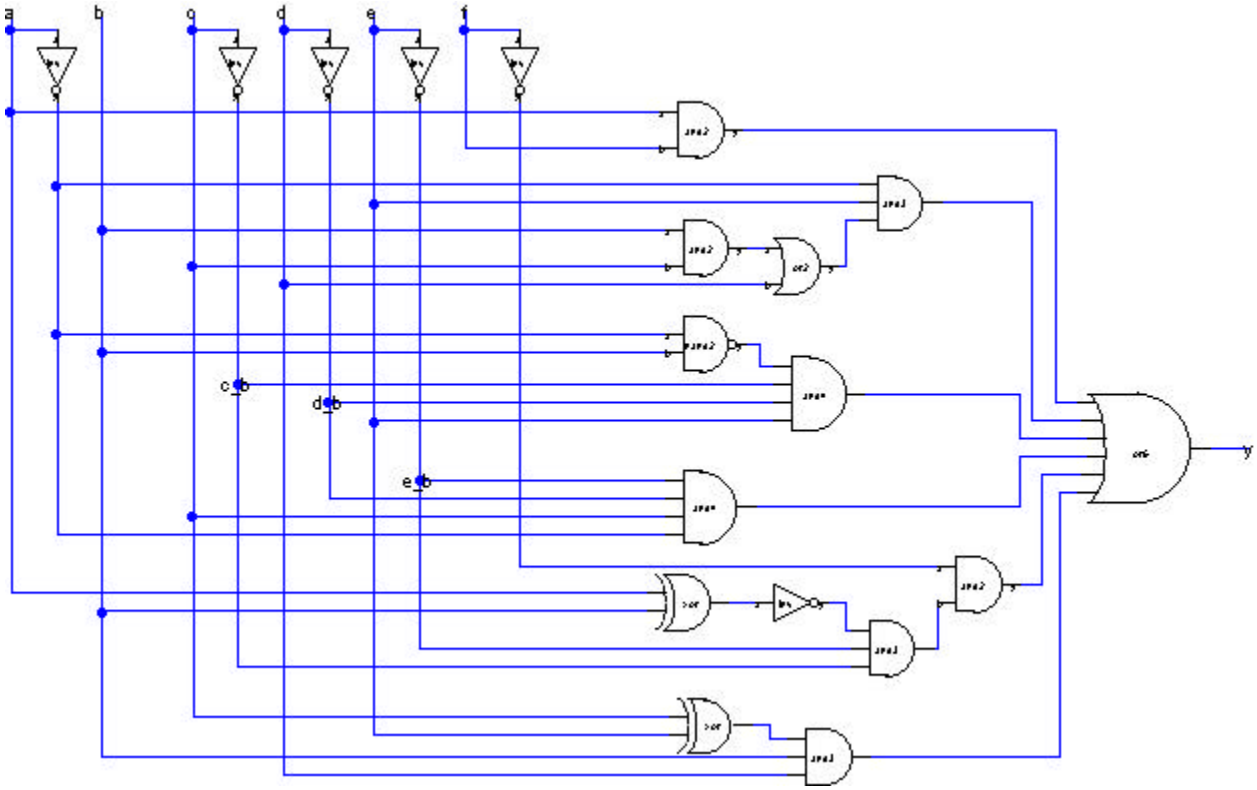
Code_Gen



bits1_36

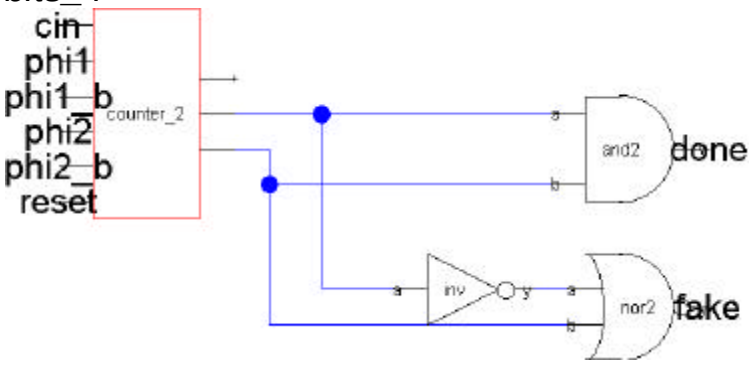


36bits

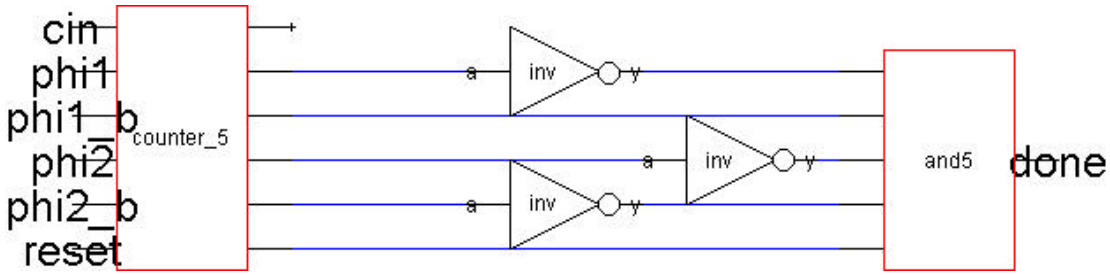


bits_F7B

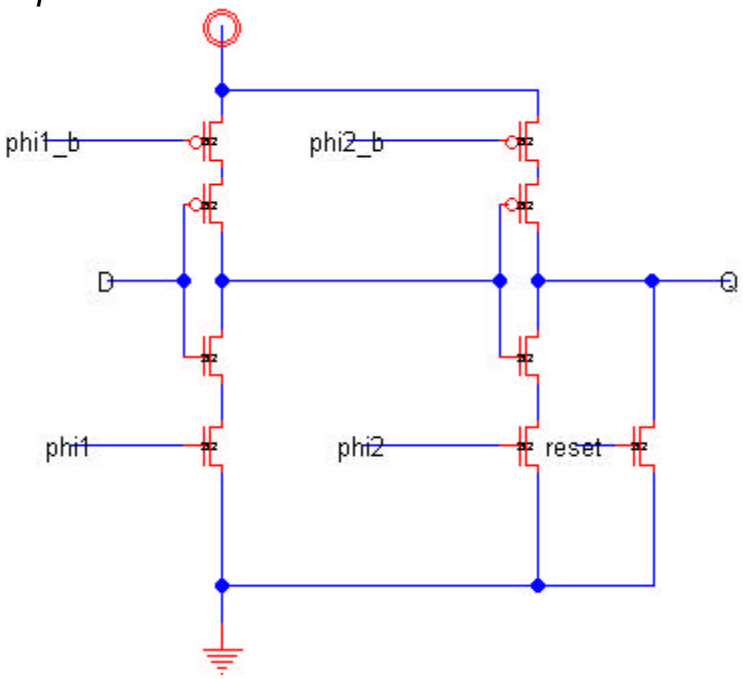
bits_4



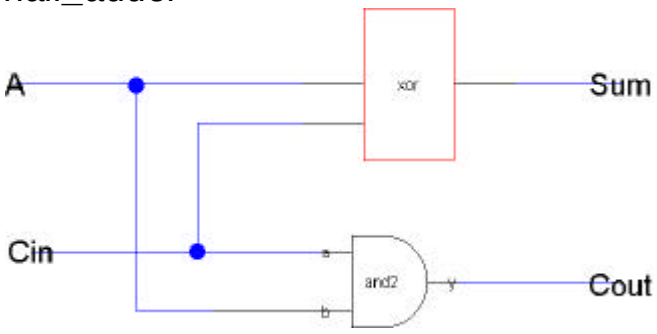
wait



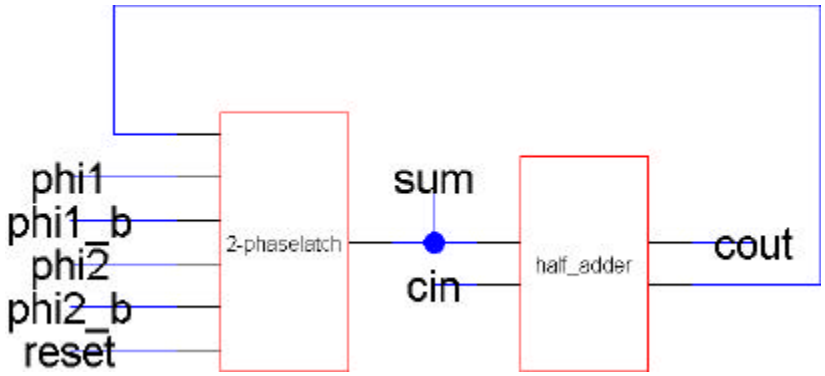
2-phaselatch



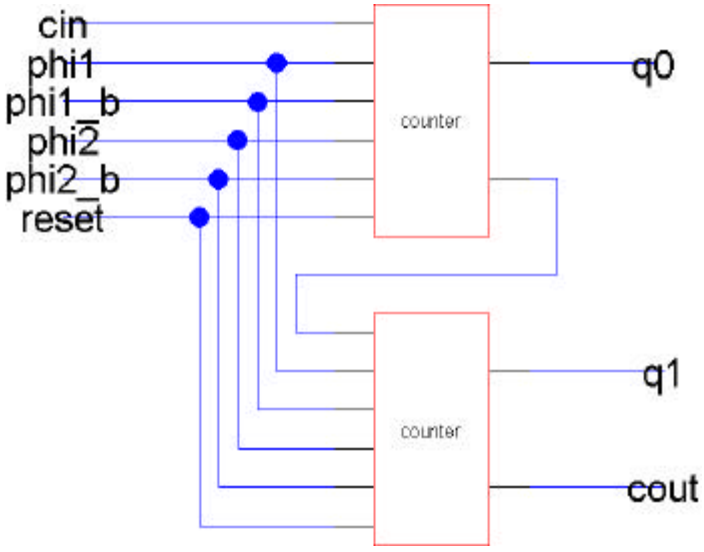
half_adder



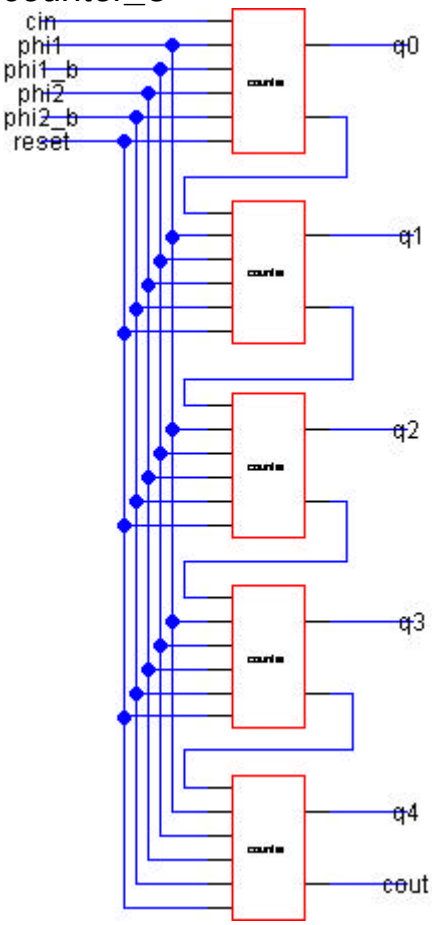
counter



counter_2

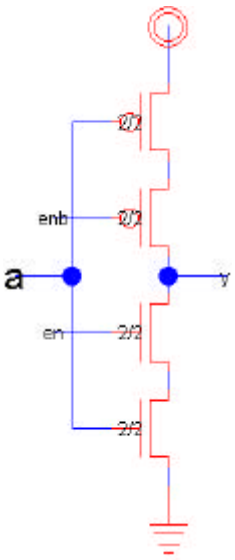


counter_5

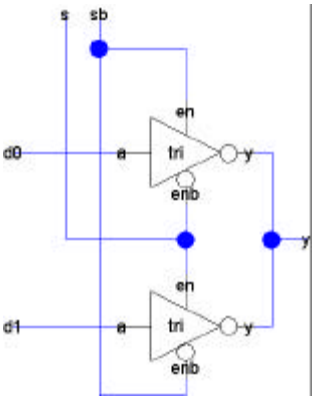


The schematics for counter_6, counter_8, and counter_13 are not shown due to their repetitive nature, but are identical in arrangement to counter_5.

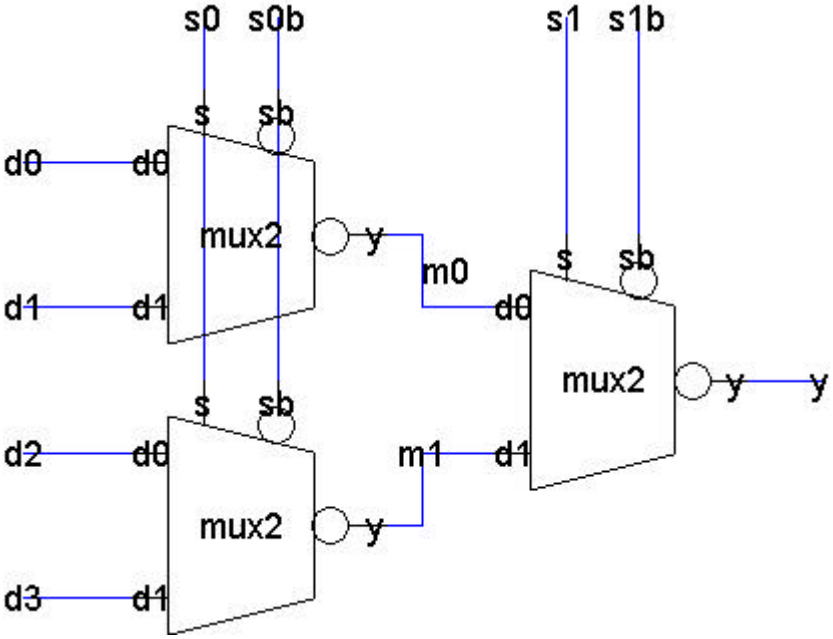
tri



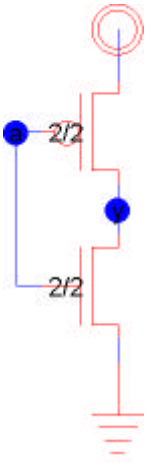
mux2



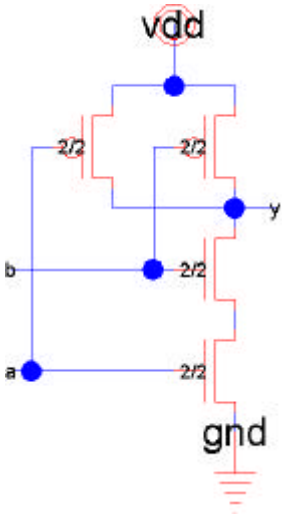
mux4



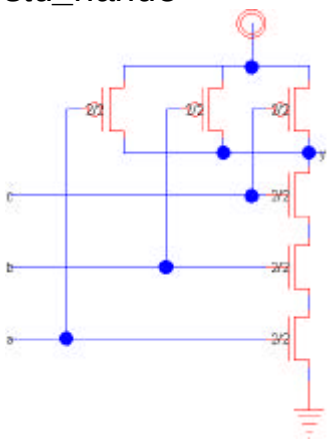
inv



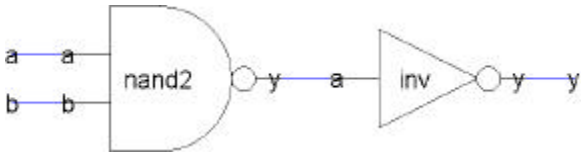
nand2



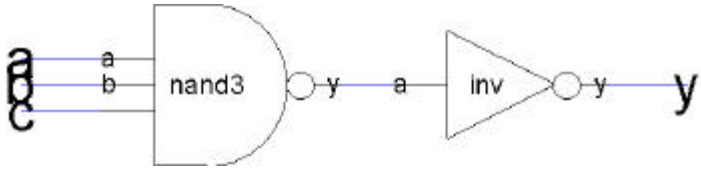
std_nand3



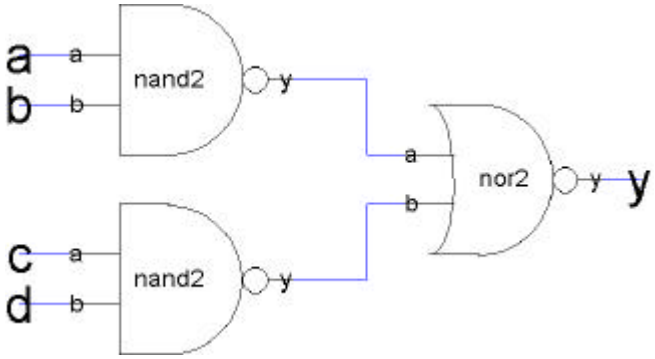
and2



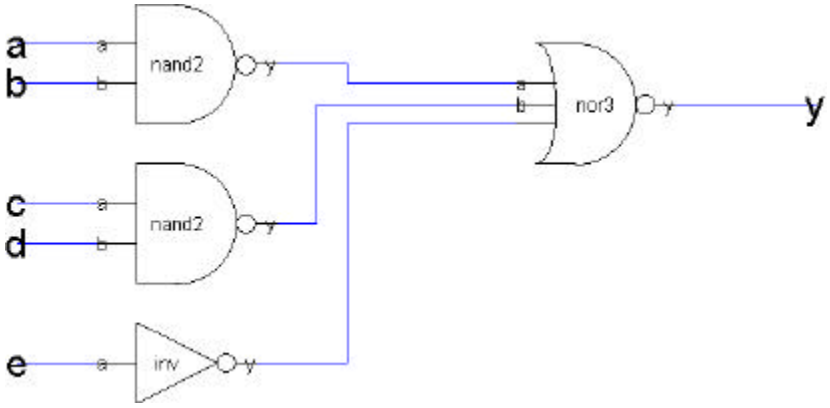
and3



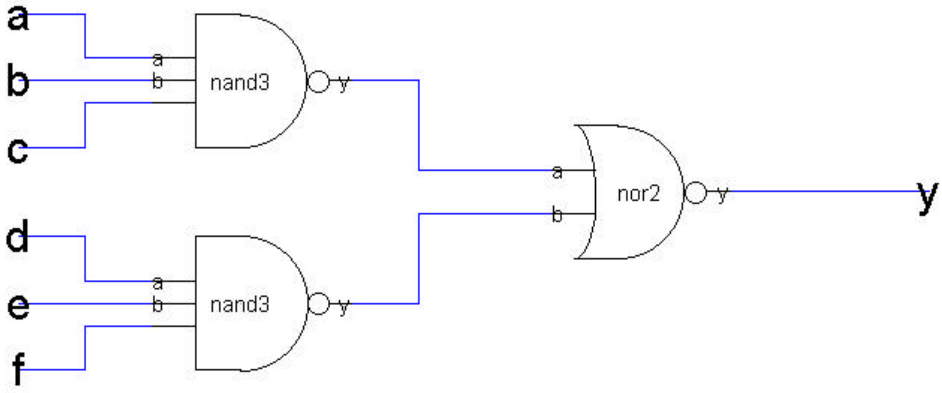
and4



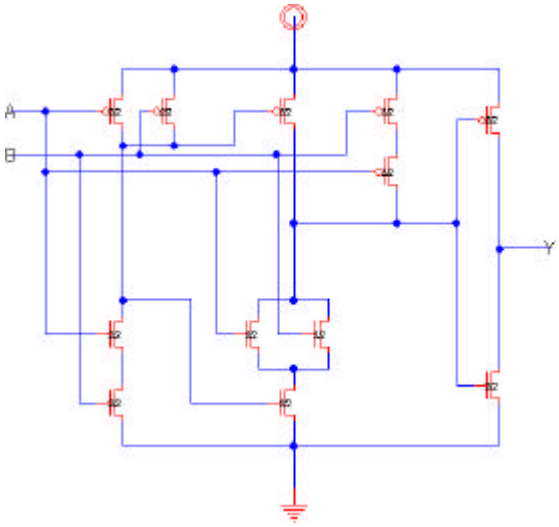
and5



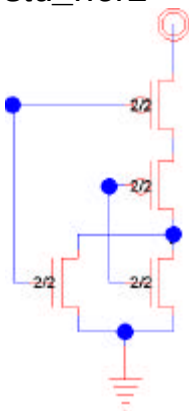
and6



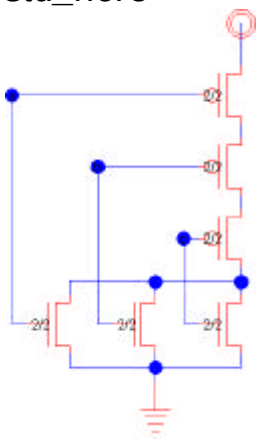
xor2



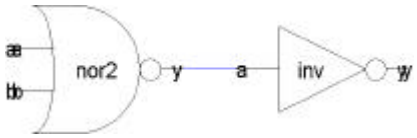
std_nor2



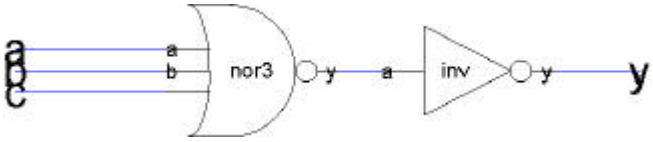
std_nor3



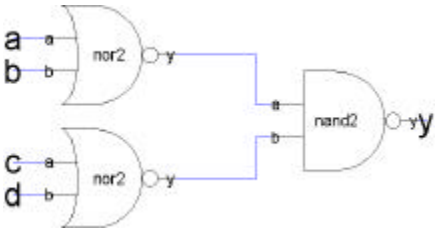
or2



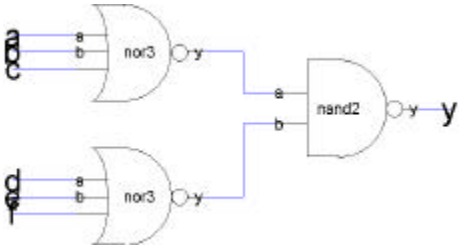
or3


a
b
c
a
b
nor3
y
a
inv
y
y

or4

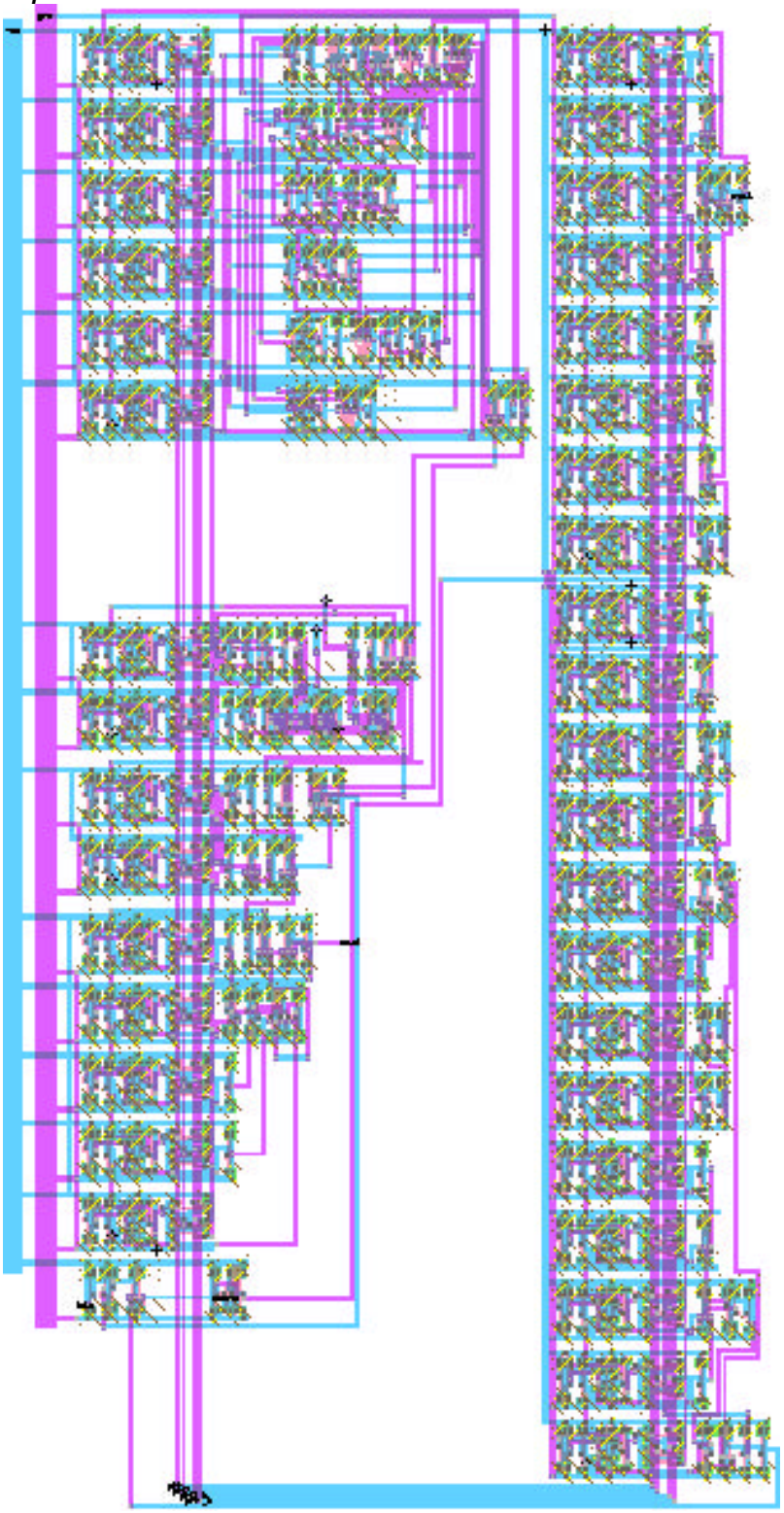

a
b
a
b
nor2
y
c
d
a
b
nor2
y
a
b
nand2
y
y

or6

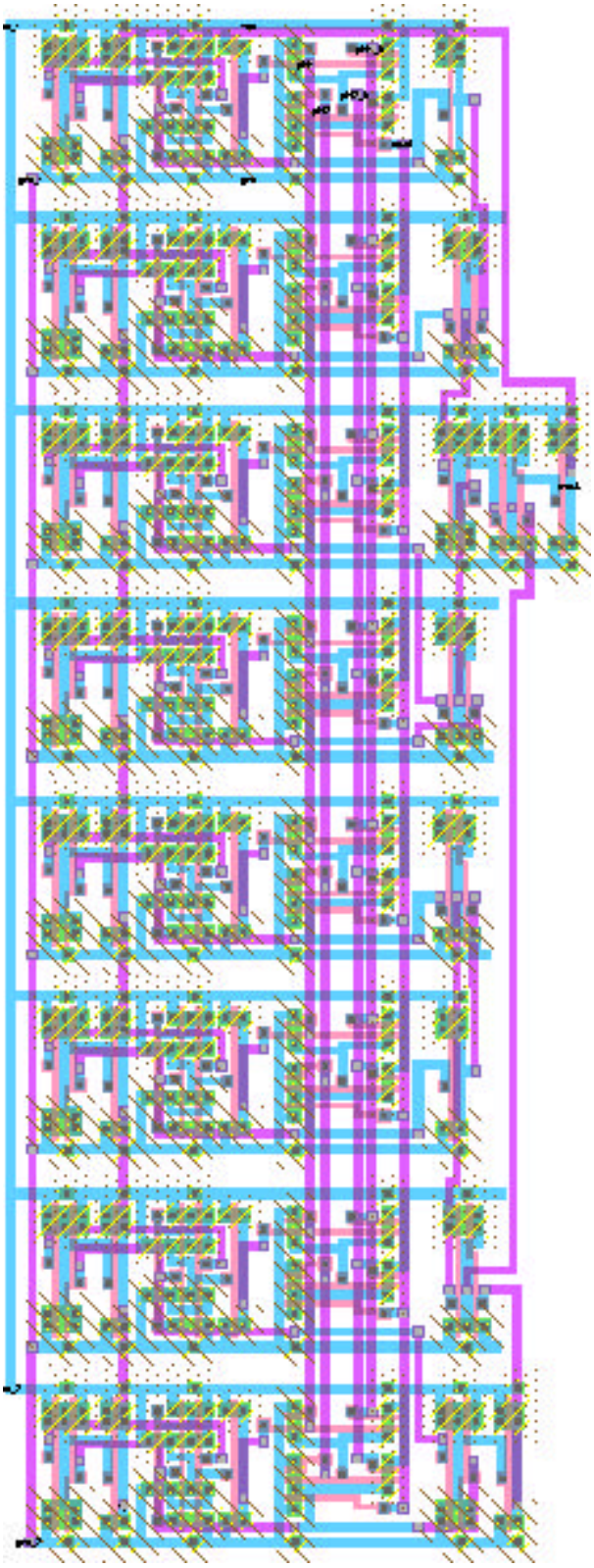

a
b
c
a
b
c
nor3
y
d
e
f
a
b
c
nor3
y
a
b
nand2
y
y

Layout

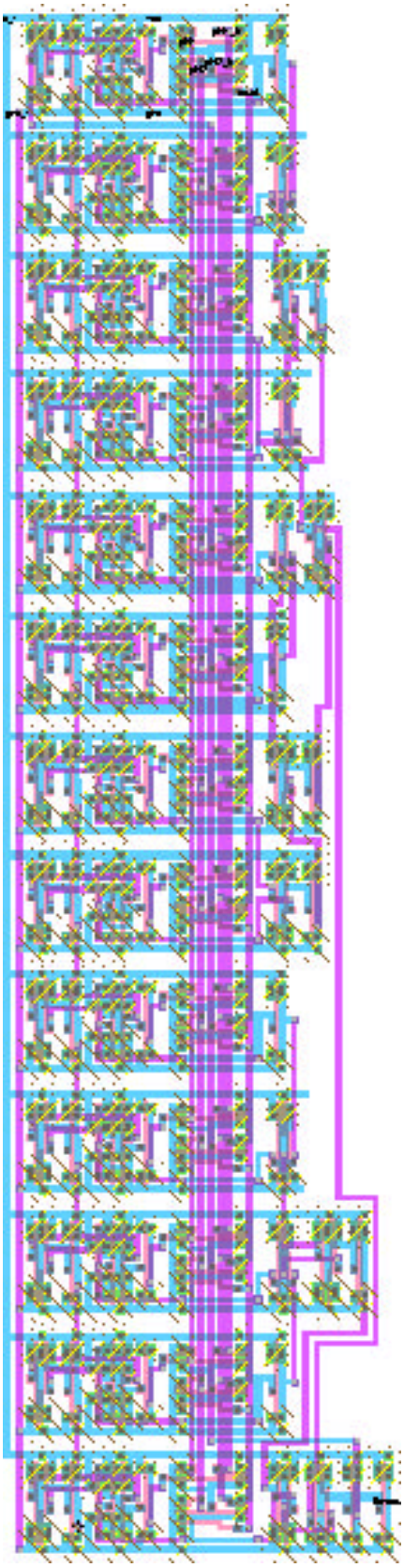
top



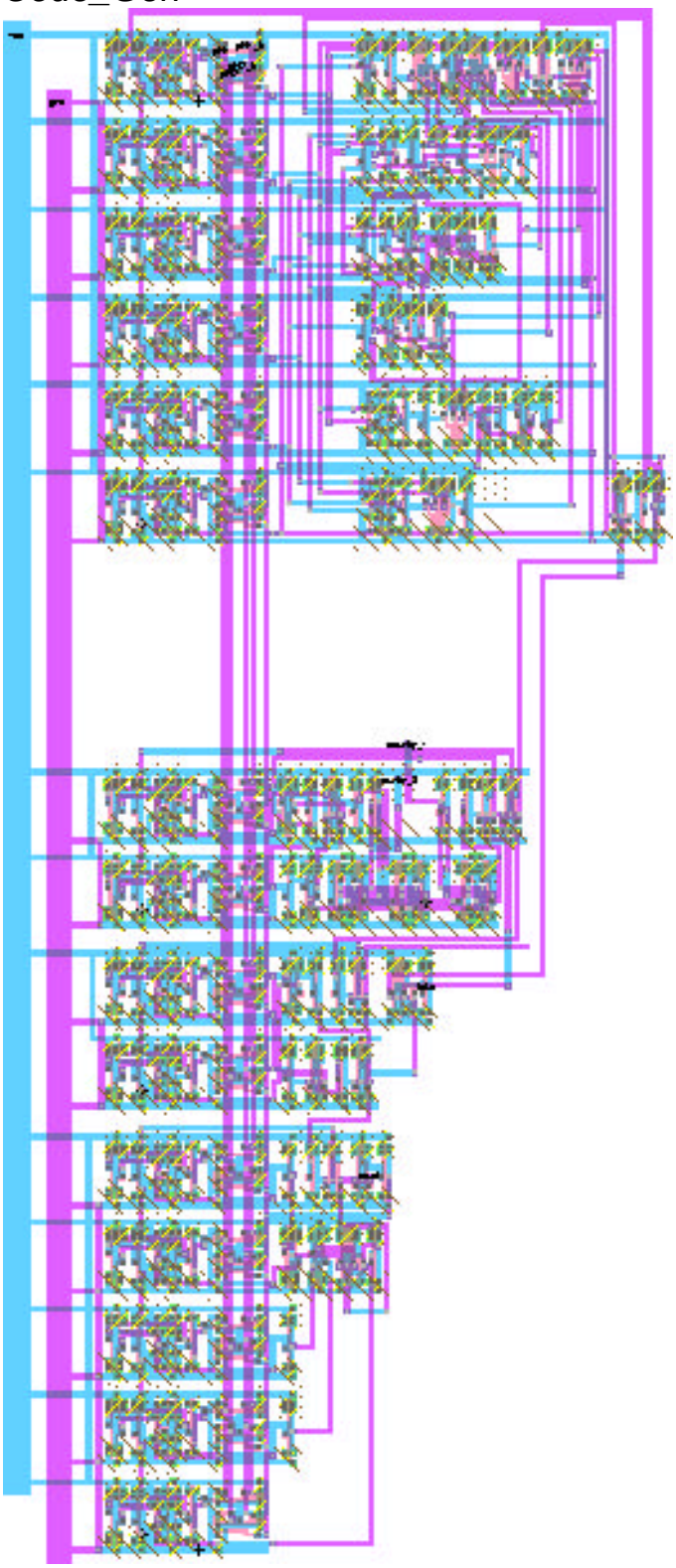
mask



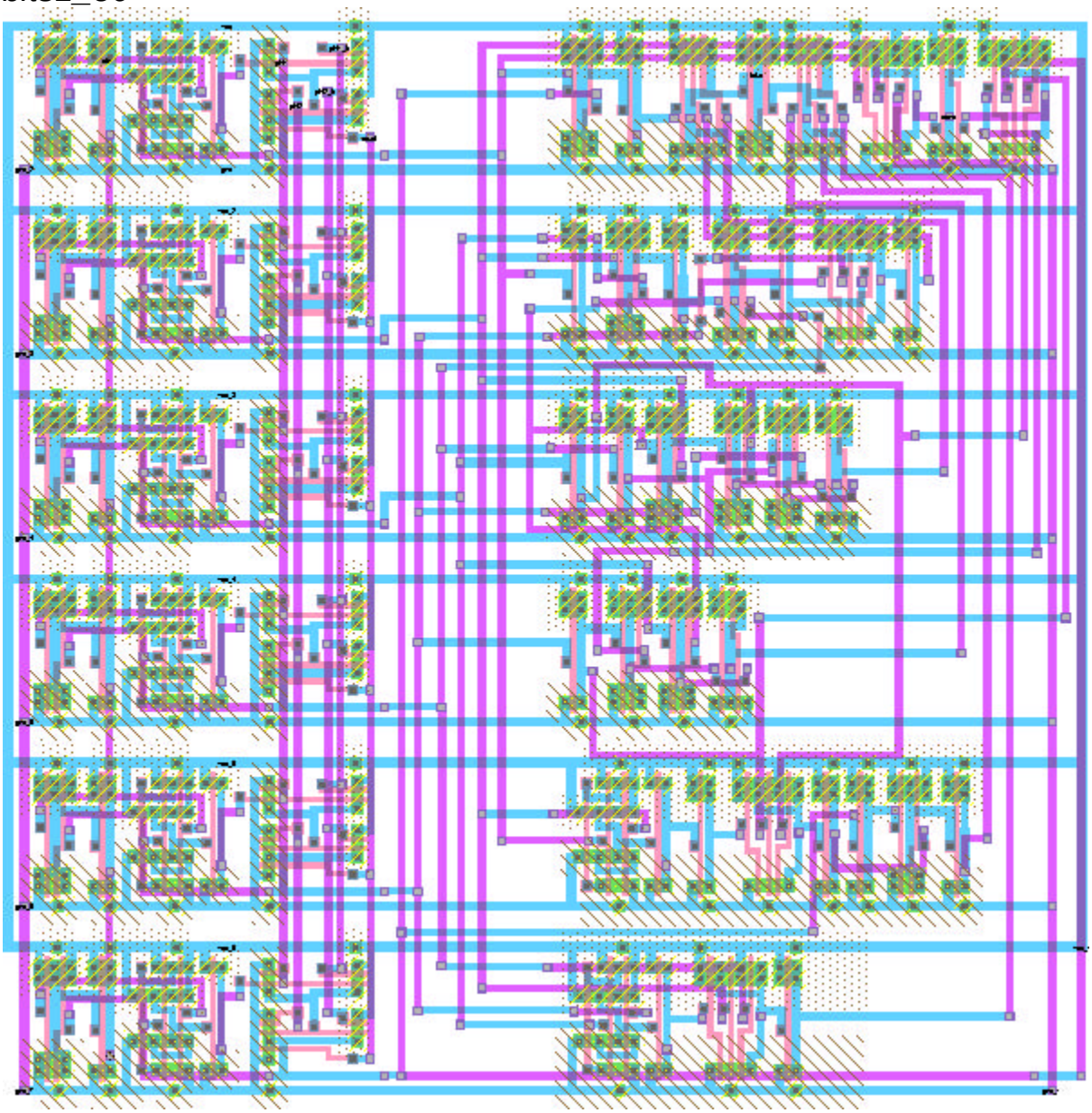
timeout



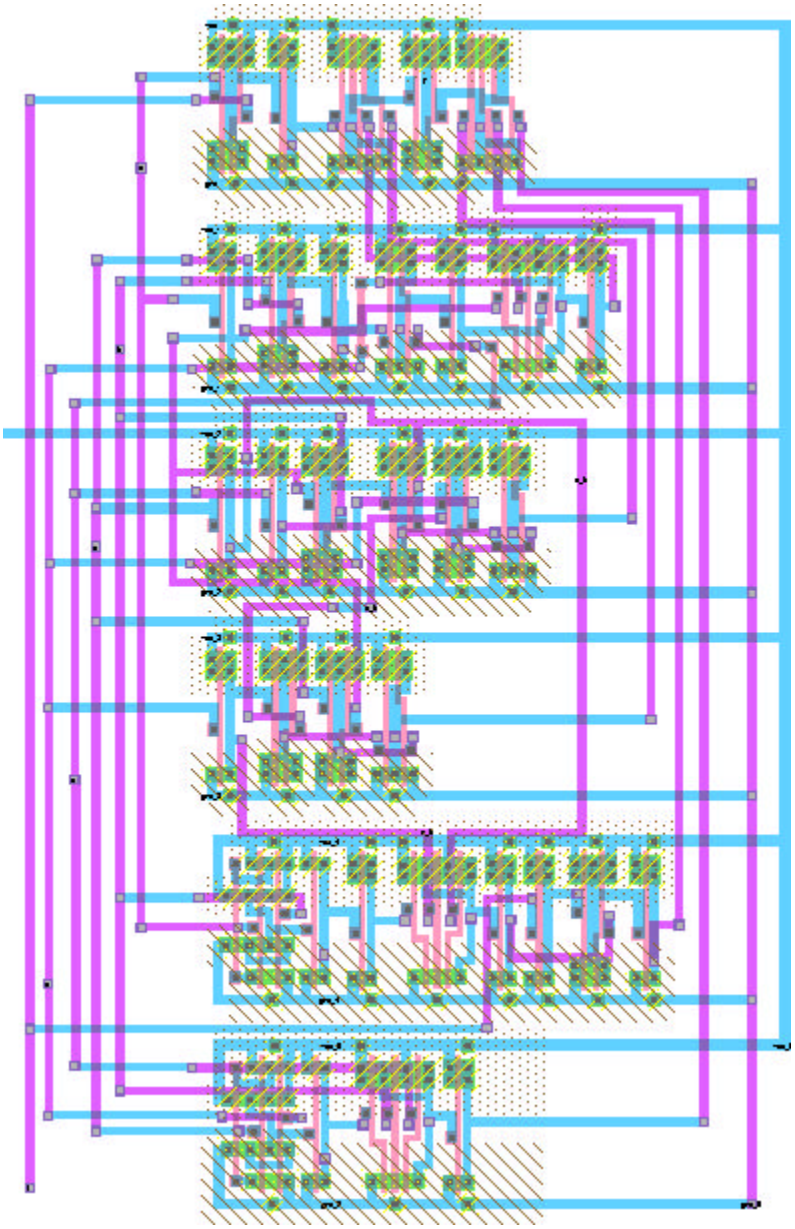
Code_Gen



bits1_36

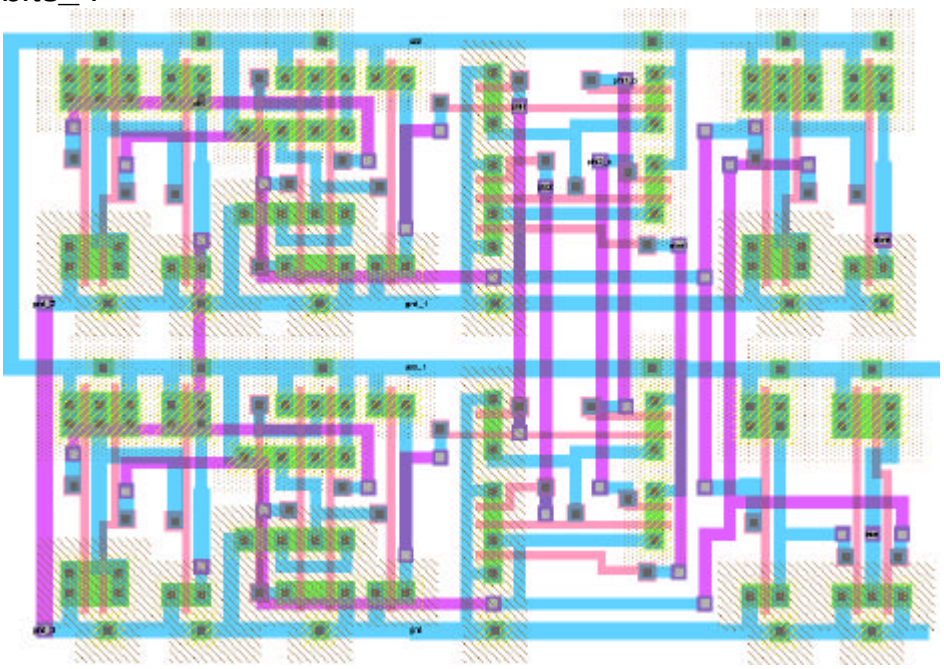


36bits

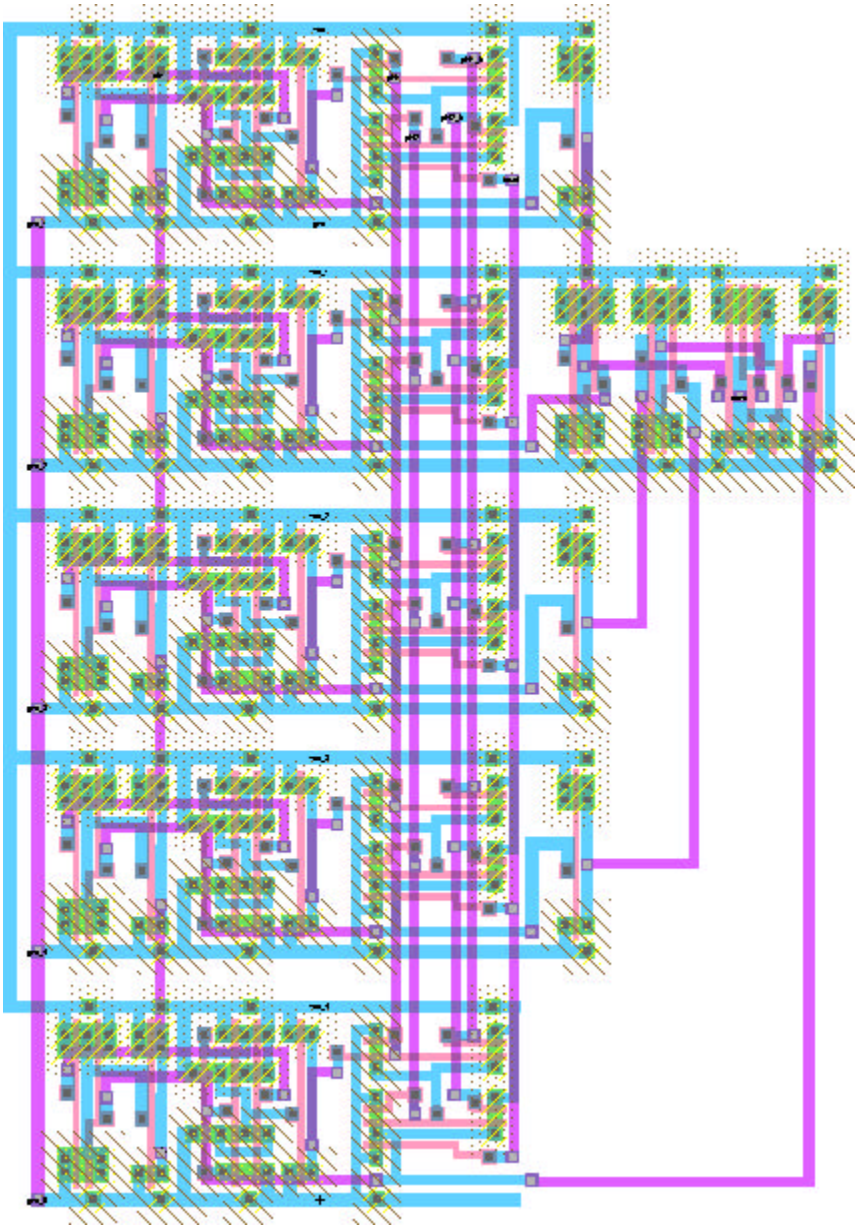


bits_F7B

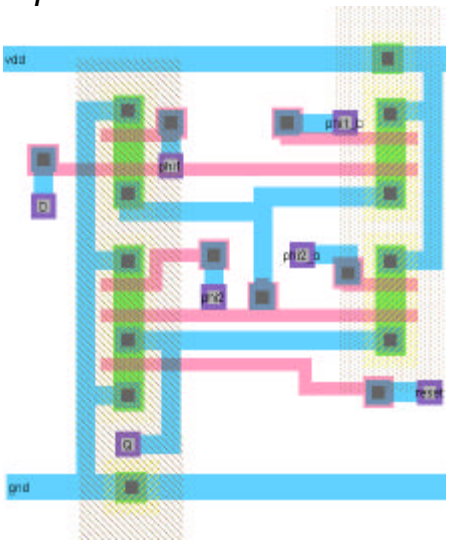
bits_4



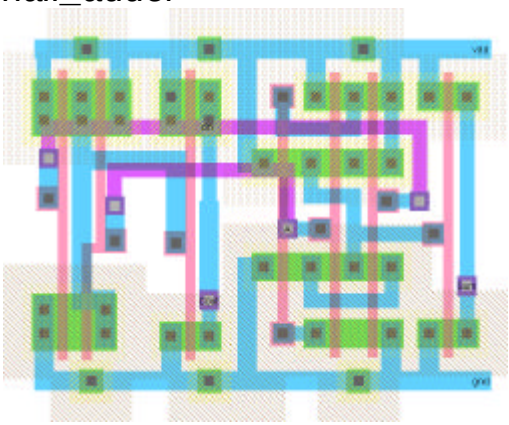
wait



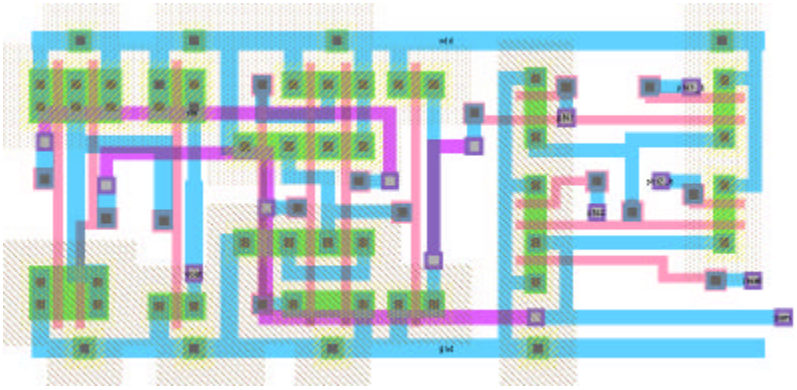
2-phaselatch



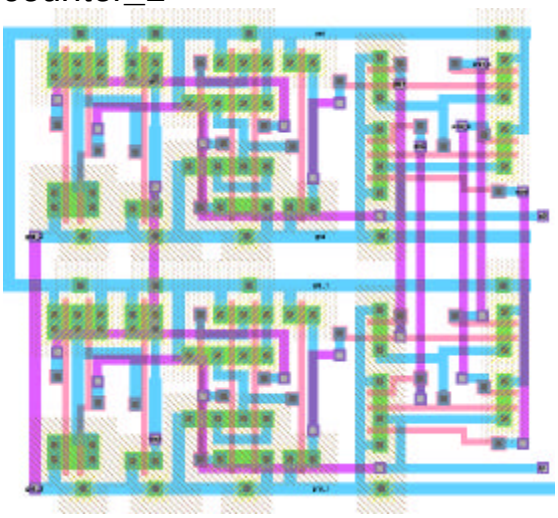
half_adder



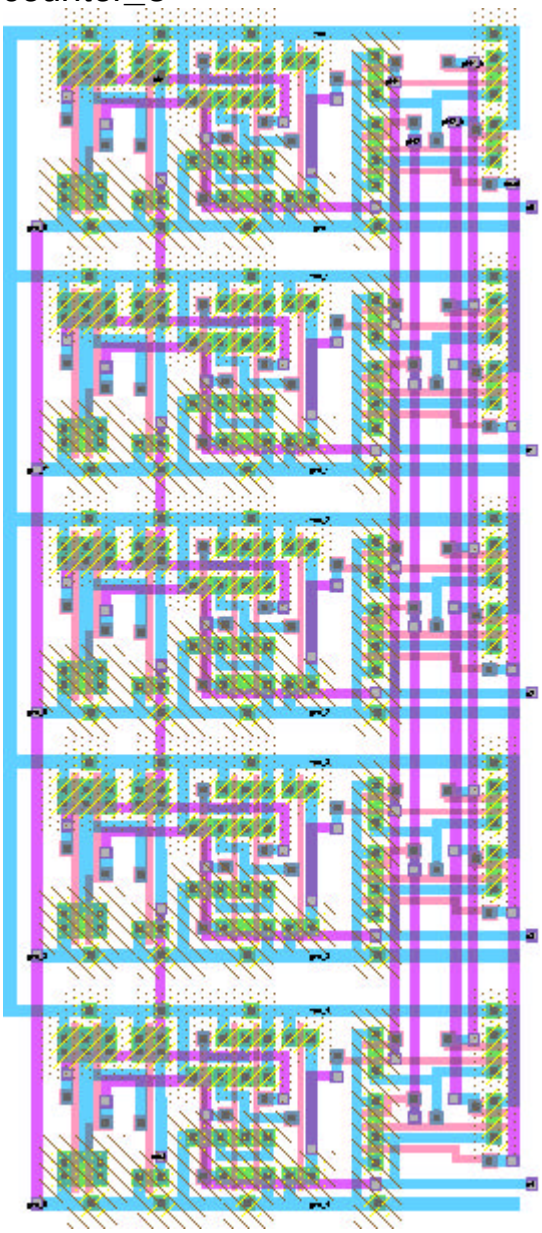
counter



counter_2

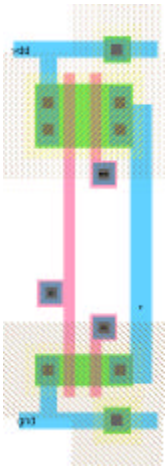


counter_5

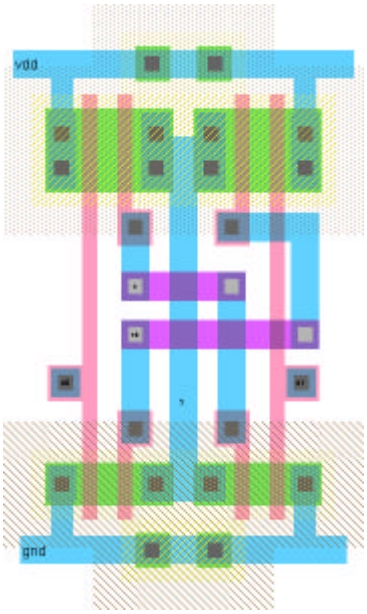


The layouts for counter_6, counter_8, and counter_13 are not shown due to their repetitive nature, but are identical in arrangement to counter_5.

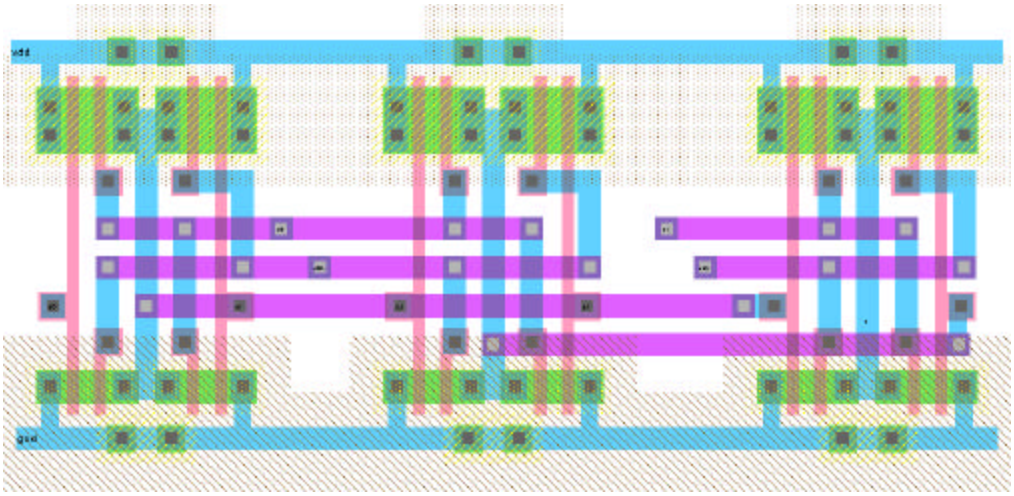
tri



mux2

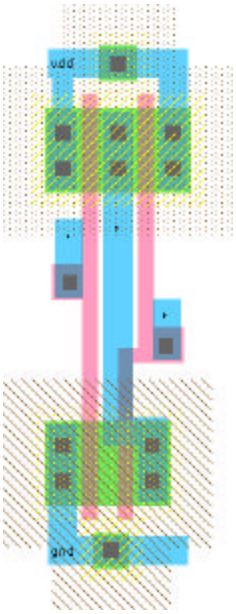


mux4

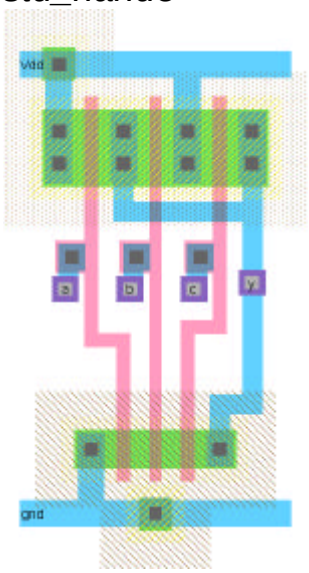


inv

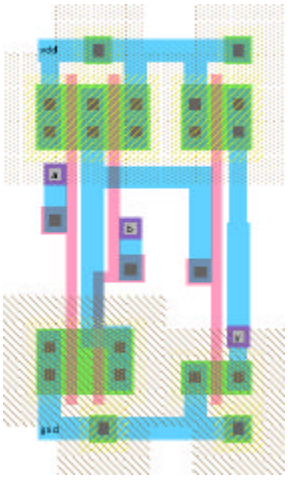
nand2



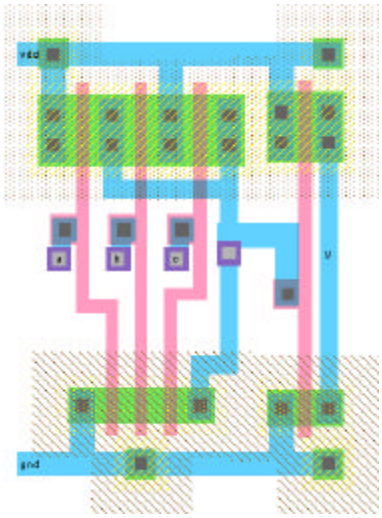
std_nand3



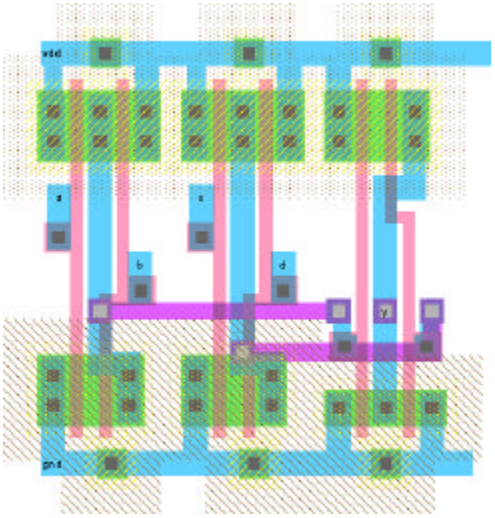
and2



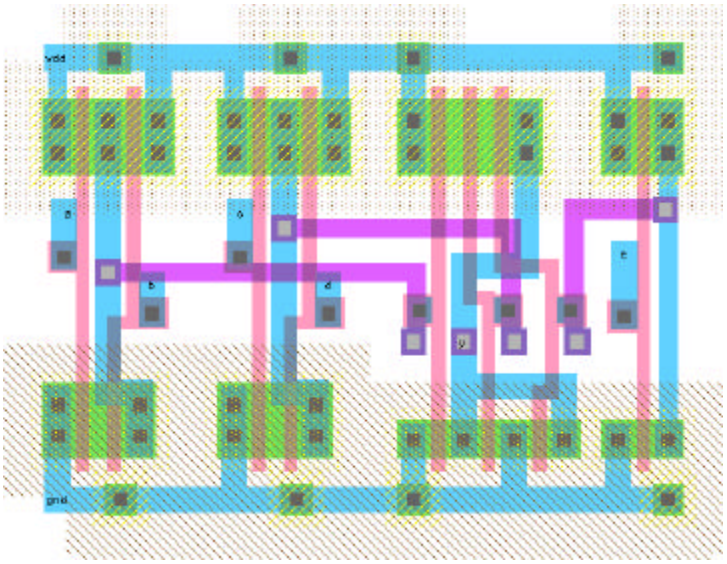
and3



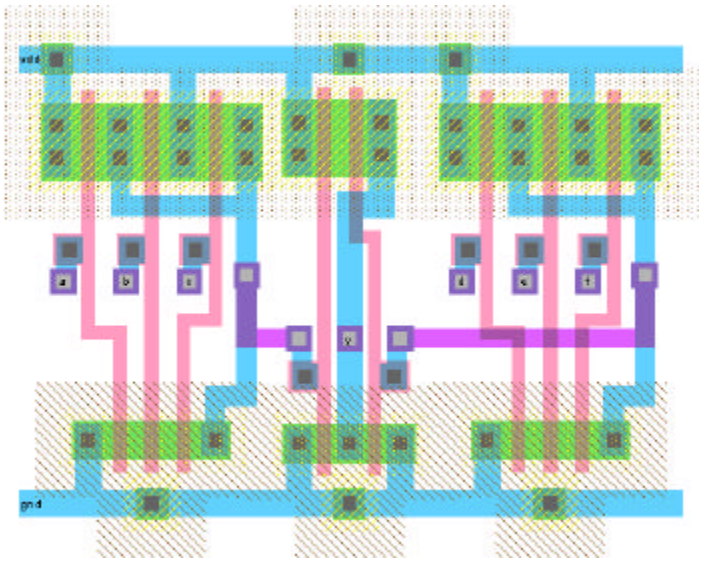
and4



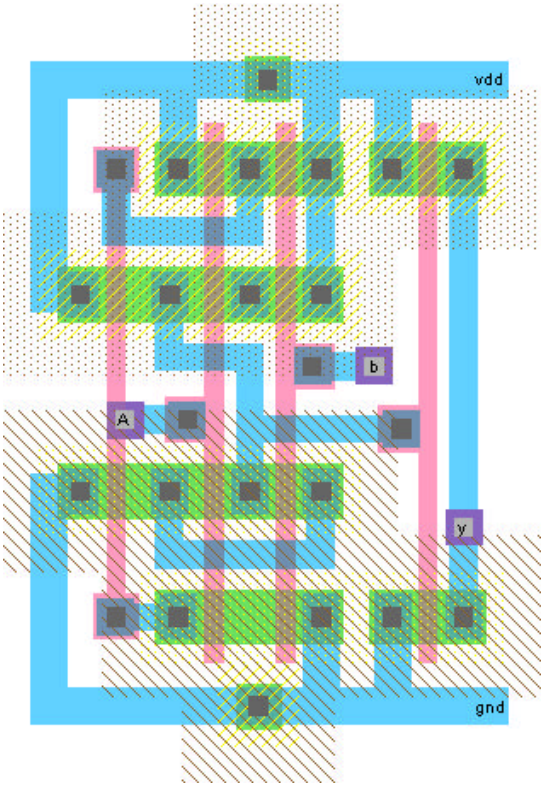
and5



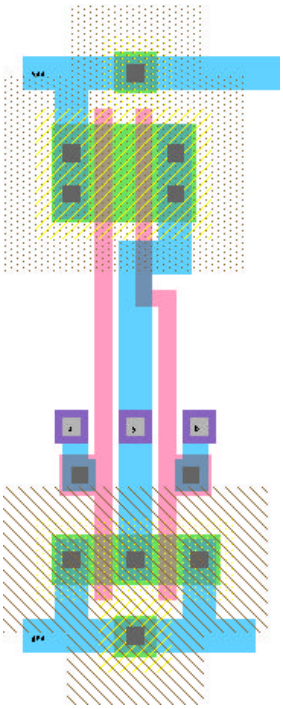
and6



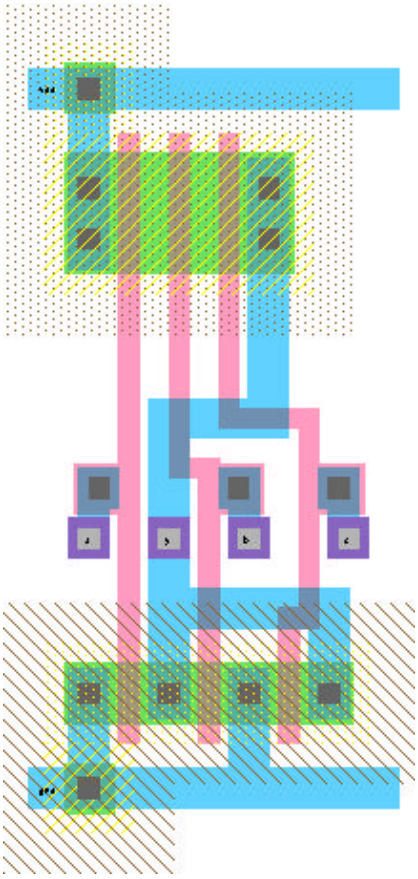
xor2



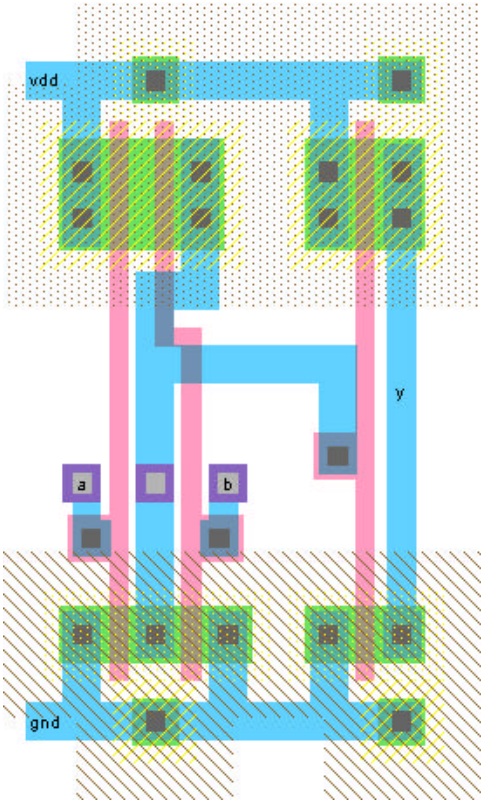
std_nor2



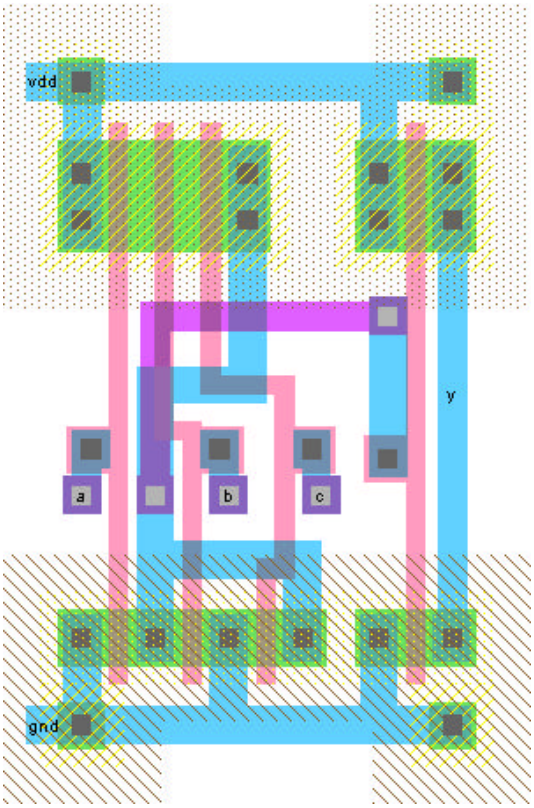
std_nor3



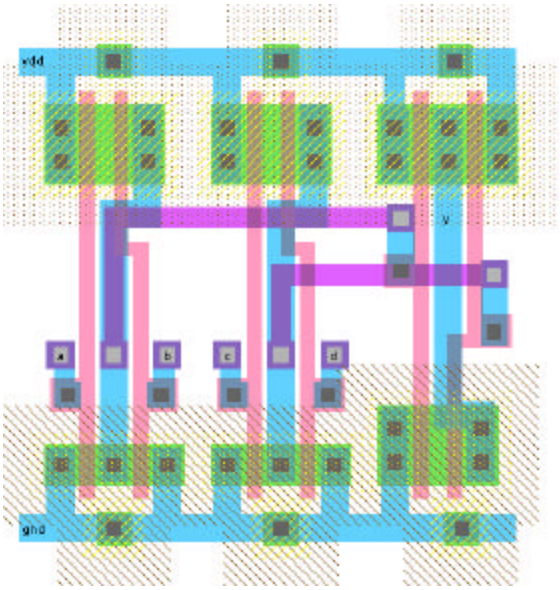
or2



or3



or4



or6